Autoencoder Applications

AW

Lecture Overview

1. Recap

2. Recommender Systems

3. Word2Vec

Matrix Factorization (MF)

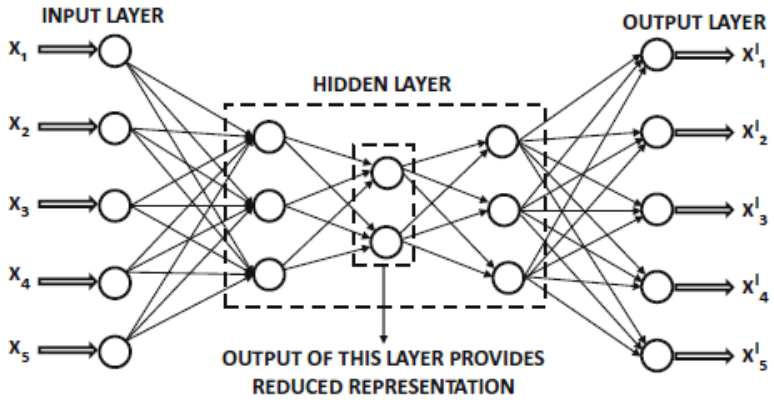
- $n \times d$ matrix data D (n inputs, d features) factorized into $D = UV^T$ where U has dimension $n \times p$ is a representation of data
- *kernel matrix factorization* nonlinear autoencoders:

Example. Shallow feature extraction NN:

- hidden layer with sigmoid and output linear.
- Input-to-hidden matrix W^T; hidden-to-output V^T
- Then output of hidden layer is $U = \text{sigmoid}(DW^T)$ notice that because sigmoid is applied elementwise U is a matrix
- Just like in linear case, if we use the mean-squared error as the loss function, we are optimizing $\|D - UV^T\|_F^2$ over the entire training data.
- So by training we get factorization $D \approx UV^T$

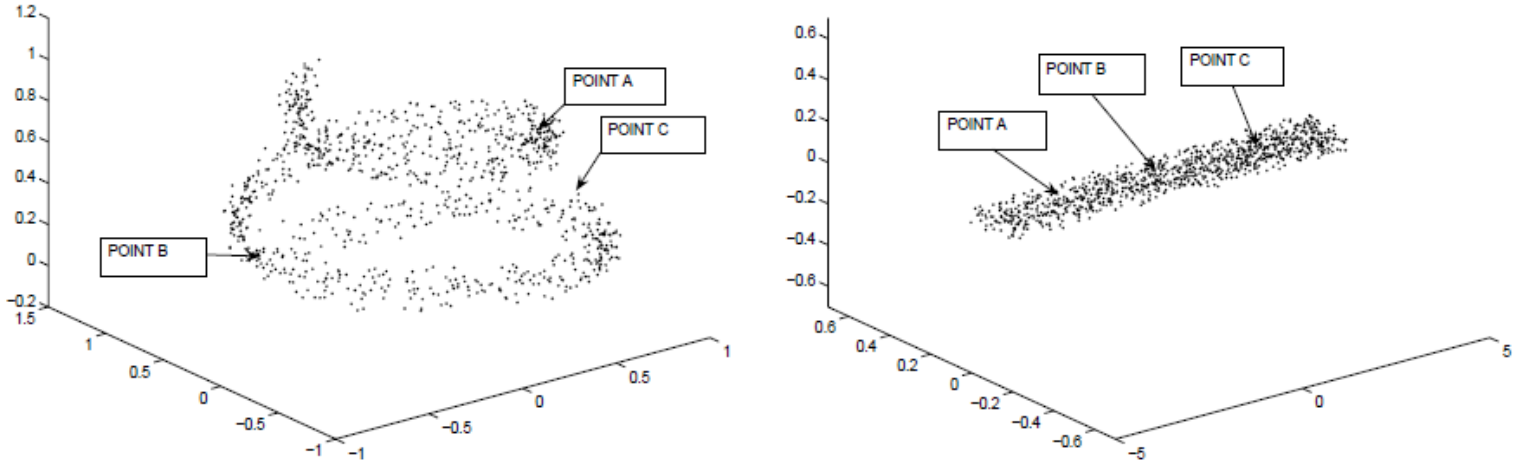
Deep encoders

- Example is simplistic compared to what is considered typical in kernel methods, in reality multiple hidden layers are used to learn more complex forms of nonlinear dimensionality reduction



- Deep non-linear encoders can achieve reductions that are not possible by linear methods such as SVD and PCA

Non-Linear Autoencoders



- The multiple layers provide *hierarchically* reduced representations of the data.
  - For some data domains like images, hierarchically reduced representations are particularly natural.
- Just like PCA nonlinear dimensionality reduction is also a form of manifold learning but it might map a manifold of arbitrary shape into a reduced representation.
  - Extreme reductions are often achieved e.g. 784 dimensional data to 6 dimensional with images of handwriting

Lecture Overview

1. Recap

2. Recommender Systems

3. Word2Vec

Recommender System (RS): the Idea

- In recommender systems (RS), we have an $n \times d$ ratings matrix D with n users and d items.
- Most of the entries in the matrix are unobserved. Given an entry (i, j) need to do predictions for the value of D_{ij}.

Method:

1. Compute user embedding (representation by a vector in $\mathbb{R}^t$ for some $t < d$)
2. Compute item embedding into the same space
3. Compute prediction as dot product

- Use factorizations of D as embeddings

Recommender System: Idea Formalization

- Let $\hat{R}$ be matrix of predicted values, let $t \times n$ matrix A be user embeddings and let $t \times d$ matrix B be item embeddings, let also b_u be user specific bias for user u (for all items) and c_i be item specific bias of item i
- Then $\hat{R} = A^T B + \mathbb{I}\vec{b} + \vec{c}^T\mathbb{I}$ where $\mathbb{I}$ is a matrix with every entry 1. Loss function for prediction is $\left\|D - \hat{R}\right\|_F^2$
- Embeddings are matrices U for users and V for items obtained by D's approximate SVDs UV^T (orthogonal columns in both) where $U = DW^T$ is $n \times t$ matrix and V^T orthogonal $t \times d$ matrix obtained by tying weights in W and V^T.
- But where do we get these when D is not known?

Difficulties with Autoencoder

- If some of the inputs are missing, then using an autoencoder architecture will implicitly assume default values for some inputs (like zero).
  - This is a solution used in some recent methods like *AutoRec*.
  - Does not exactly simulate classical Matrix Factorization (MF) method used in recommender systems because it implicitly makes assumptions about unobserved entries.
- None of the proposed architectures for recommender systems in the deep learning literature exactly map to the classical factorization method of recommender systems.

How to Handle Missing Entries

- Recommender systems are well suited to *elementwise* learning, that heavily depends on representation:
- when a small subset of values (ratings) from an row (item is available we may train the network on inputs that are triplets of the form (RowId, ColumnId , Rating)
  - We can use one hot encoding.
  - Rating for every user/movie is not necessary on a trained network

Example:

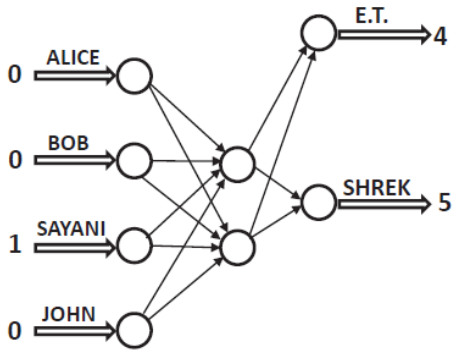
Suppose we want have 4 customers Alice, Bob, Sayani and John whose ratings on movies “Shrek”, “ET”, “Nixon” and “Ghandi” we are trying to predict

How to Handle Missing Entries

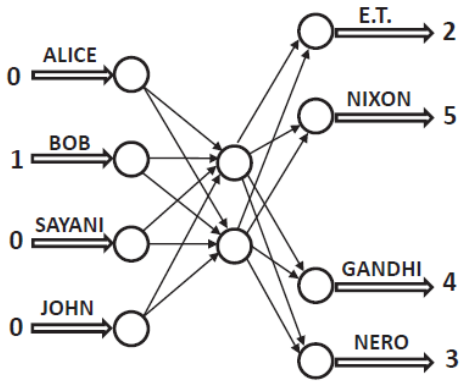
Example:

Suppose we want have 4 customers Alice, Bob, Sayani and John whose ratings on movies “Shrek”, “ET”, “Nixon” and “Ghandi” we are trying to predict

- Say Bob has ratings for all 4 and Sayani only on “ET and Shrek”. How do we predict ratings of Sayani rating on “Nixon”?
- Train network on Bob and then apply to Sayani setting inputs to random



OBSERVED RATINGS (SAYANI): E.T., SHREK



OBSERVED RATINGS (BOB): E.T., NIXON, GANDHI, NERO

Classical MF vs RS NN

- Autoencoders map row values to row values.
- Each user has his/her own neural architecture with missing outputs.
- Weights across different user architectures are shared.
    - Not standard definition of autoencoder.
    - Can handle incomplete values but cannot handle out-of-sample data.
- Since the two weight matrices are U and V^T, the one-hot input encoding will pull out the relevant row from UV^T.
- Since the outputs only contain the observed values, we are optimizing the sum-of-square errors over observed values.
- Objective functions in both MF and RSNN are equivalent!

Training Equivalence

- For k hidden nodes in one hidden layer, there are k paths between each user and each item identifier.
- Backpropagation updates weights along all k paths from each observed item rating to the user identifier.
- These k updates can be shown to be *identical* to classical matrix factorization updates with stochastic gradient descent.
- Backpropagation on neural architecture is identical to classical stochastic gradient descent.

Advantage of RS NN view over classic MF

- The neural view provides natural ways to add power to the architecture with nonlinearity and depth.
  - Much like a child playing with a LEGO toy.
  - You are shielded from the ugly details of training by an inherent modularity in neural architectures.
  - The reason of this magical modularity is backpropagation.
- If you have binary data, you can add logistic outputs for logistic matrix factorization.
- Several MF methods in machine learning can be expressed as row-index-to-row-value autoencoders (but not widely recognized–RS matrix factorization a notable example)
- Several row-index-to-row-value architectures in NN literature are also not fully recognized as matrix factorization methods
  - Example: word2vec (in this lecture- later)

Lecture Overview

1. Nonlinear and Deep Encoders

2. Recommender Systems

3. Word2Vec

Word2Vec idea

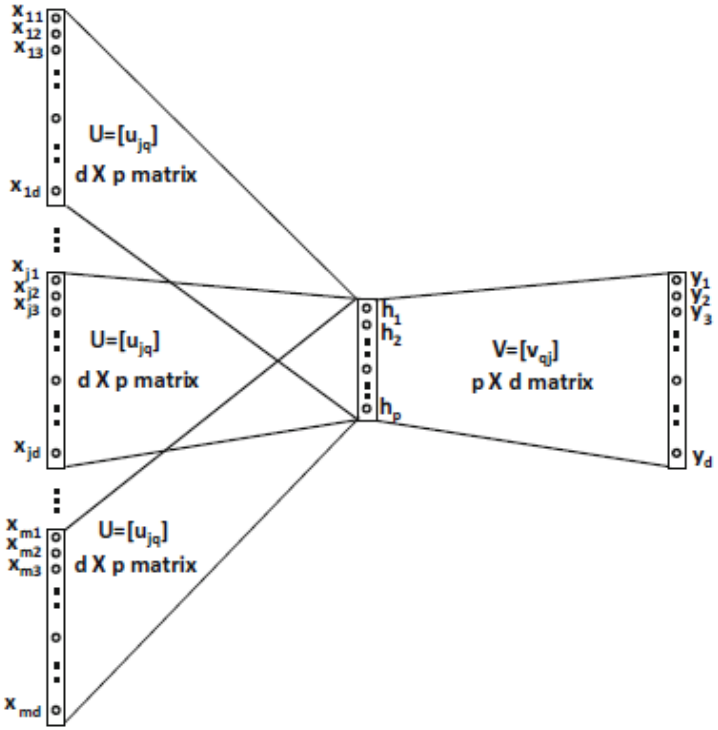
- *Word2vec* computes embeddings of words using sequential proximity in sentences.
  - If *Paris* is closely related to *France*, then *Paris* and *France* must occur together in small windows of sentences.
    - Their embeddings into some common space (usually as vectors in $\mathbb{R}^n$) should also be somewhat similar.
- Continuous bag-of-words model predicts central word from context window.
- Skipgram model predicts context window from central word.

Continuous Bag of Words (CBOW) Model

- Context-word pairs: a window of context words is input, and a single target word is predicted
- Context: $m = 2 \cdot t$ words where $w_1, \dots w_t$, are t words before and $w_{t+1}, \dots w_m$ are t words after the target word w.
- Target word w is a categorical variable that can take d values where d is the size of the vocabulary
- The goal of the NN is to compute probability $P(w|w_1 w_2 \dots w_m)$. Embeddings are provided by hidden layer of this NN
- 1-hot encoding of the vocabulary:
  - order/enumerate all words in the vocabulary
  - to word i associate a binary d-dimensional vector that has 1 in position i and 0 everywhere else
- Context is a $d \times m$ matrix X where $x_{ij} = \begin{cases} 1 \text{ if } w_i \text{ is word } j \\ 0 \text{ otherwise} \end{cases}$

Row-Index-to-Row-value Architecture

- The hidden layer has dimensionality p i.e. contains p units $h_1, h_2, \dots, h_p$
- Weights of connections of input representatives of same word are same (shared)
  - Word i of the vocabulary has m different input representatives - row i of X
  - Weight of connection of word i to h_q is denoted u_{iq}, so input is connected to hidden layer by matrix U
- Vector $\vec{u}_j = \left(u_{j1} \dots u_{jp}\right)^T$ is p-dim Input embedding of word j over vocabulary
- Vector $\vec{h} = \left(h_1 \dots h_p\right)^T$ is an output embedding of an instance of a context



Embeddings and their Properties

- In embedding the one-hot encodings of the input words are aggregated:

$$\vec{h} = \sum_{i=1}^{m} \sum_{j=1}^{d} \vec{u}_j x_{ij}$$

- Since in embedding 'words' are summed up their position doesn't matter – which is why continuous 'bag of words'
- Some sequential information is preserved because it is determined by window size
- Embedding of words $\vec{u}_j$ is often used as pre-training on big lexicons
- Embedding of context $\vec{h}$ is used for training dense FF NN with softmax prediction of a word in this NN:
  - The hidden to output weights matrix V and softmax activation determine output $\hat{y}_1, \ldots, \hat{y}_d$ of probabilities of each word in a given instance of the context

Word2Vec CBOW Prediction

- Activation is softmax so $\hat{y}_j = \frac{\exp\left(\sum_{q=1}^{p} h_q v_{qj}\right)}{\sum_{k=1}^{d} \exp\left(\sum_{q=1}^{p} h_q v_{qj}\right)}$
- As always for softmax loss is cross-entropy:

$$L = -y_i \log \hat{y}_i$$

- Updates are computed by backpropagation; can be easily expressed in terms of signed error $\epsilon_i = y_i - \hat{y}_i$ where

$$y_i = \begin{cases} 1 \text{ if word } i \text{ occured} \\ 0 \quad \text{otherwise} \end{cases}$$ as

$\vec{u}_i = \vec{u}_i + \alpha \sum_{j=1}^{d} \epsilon_j \vec{v}_j$ and $\vec{v}_j = \vec{v}_j + \alpha \epsilon_j \vec{h}$ where α is learning hyperparameter

- The training examples of context-target pairs are presented one by one, and the weights are trained to convergence

Skip-gram Model

- This model tries to predict the context $w_{(i-t)}w_{i-t+1} \dots w_{i-1}w_{i+1} \dots w_{(i+t-1)}w_{i+t}$ around word w_i, given the i^{th} word in the (middle of the) sentence.
- The total number of words in the context window that we are trying to predict is $m = 2t$.
- The vocabulary size is d
- The goal of the NN is to compute the probability $P(w_1 w_2 \dots w_m | w)$. Hidden layer provides embeddings of words
- We want to find an embedding of each word.
- 1-hot encoding of the vocabulary:
  - order/enumerate all words in the vocabulary
  - to word i associate a binary d-dimensional vector that has 1 in position i and 0 everywhere else

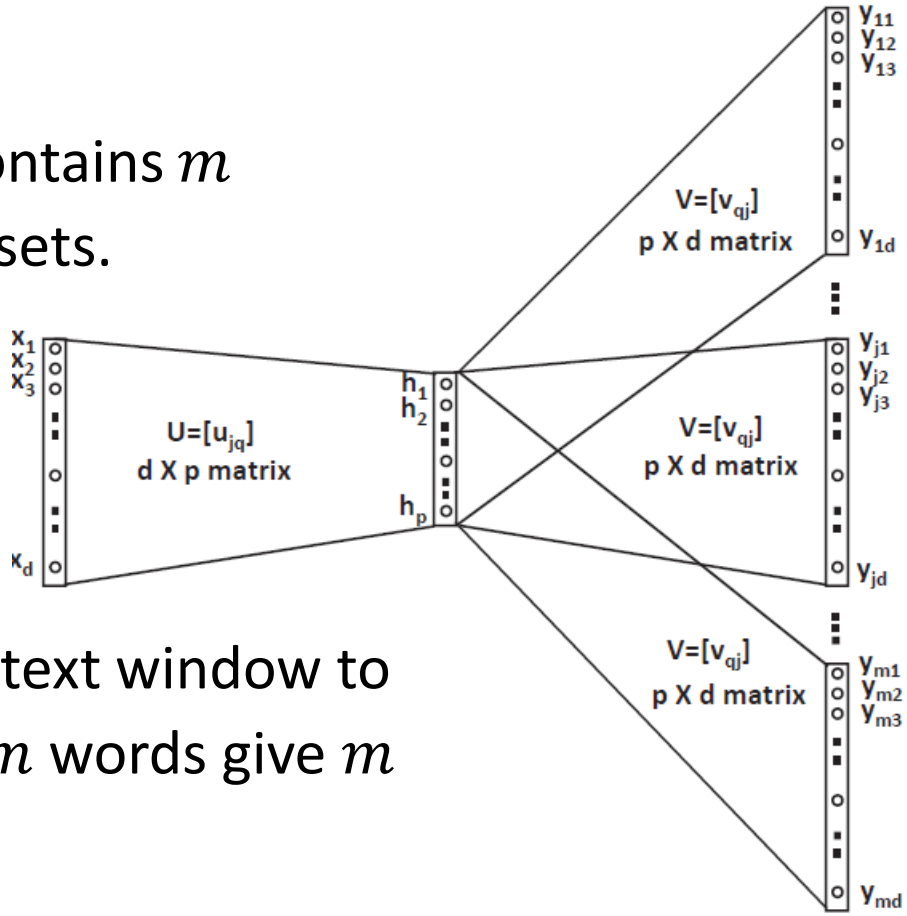
Similarities with Recommender System

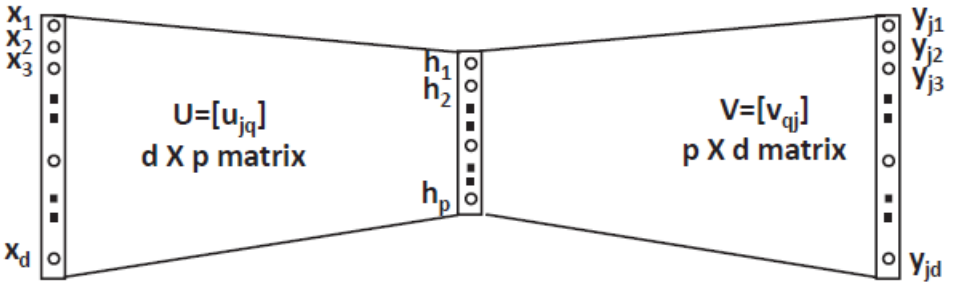
Note: One can also create a $d \times d$ word-context matrix C with frequencies c_{ij} similar to recommender systems and get embeddings from there

- Instead of user-item matrices, we have square word-context matrices.
  - The frequencies correspond to the number of times a contextual word (column id) appears for a target word (row id).
  - Analogous to the number of units bought by a user (row id) of an item (column id).
  - A fact is that skipgram *word2vec* uses an almost identical model to current recommender systems.

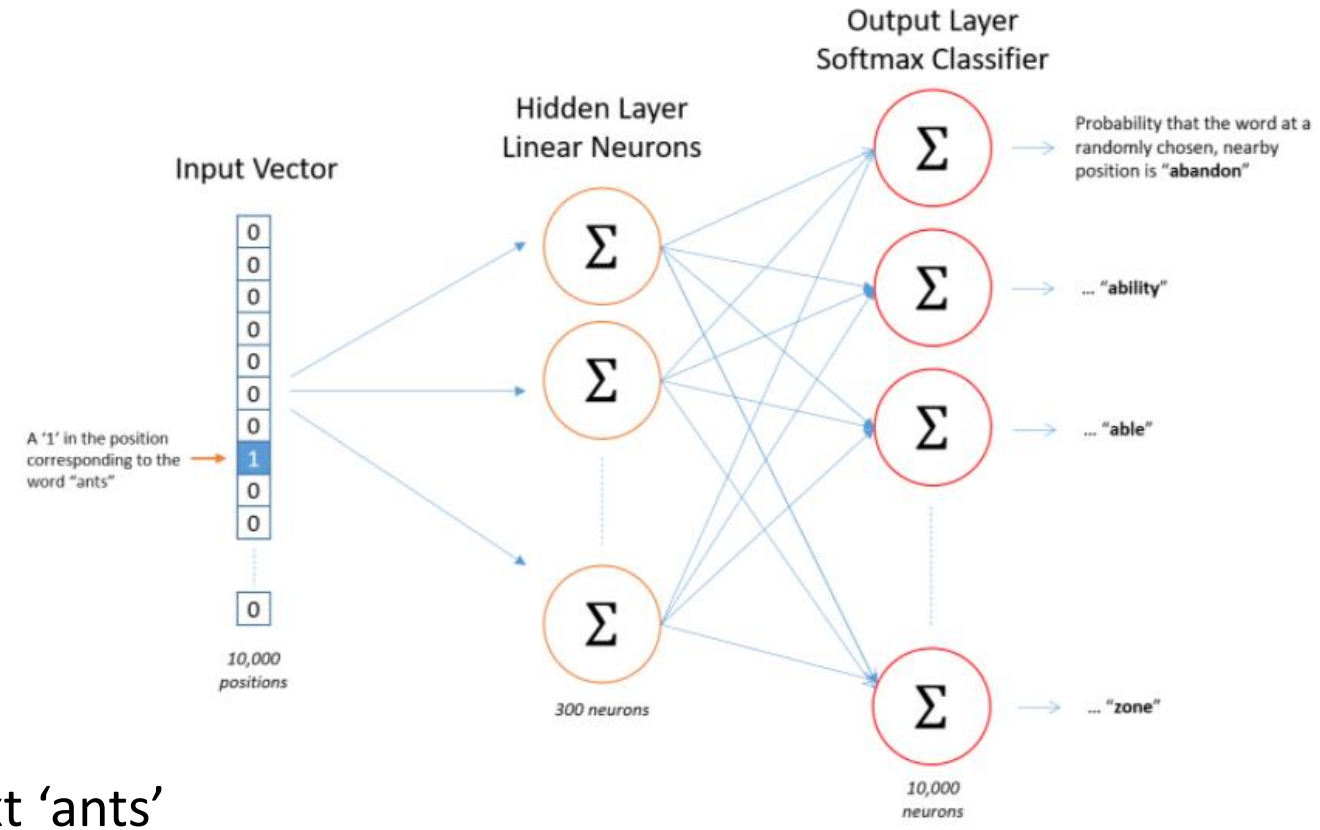
Skip-gram Architecture

- Input is the one-hot encoded word identifier and output contains m *identical* softmax probability sets.
- Since the *m* outputs are identical, we can collapse the *m* outputs into a single output.
- Mini-batch the words in a context window to achieve the same effect, i.e. m words give m d-dimensional output vectors in each context window during stochastic gradient descent





Skip-gram Architecture Example Using CORPUS



- Context 'ants'
- CORPUS has 10000 words and sentences.
- Window has $t = 5$ words so 10 words with ants at the center
- Training example: every word that is within 5 of ants in 1 hot encoding is positive example – every other word is negative example

Word2Vec Skip-Gram Embedding and Prediction

- Embedding $h_q = \sum_{j=1}^{d} u_{jq} x_j$
- Activation is softmax so $\hat{y}_{ij} = \dfrac{\exp\left(\sum_{q=1}^{p} h_q v_{qj}\right)}{\sum_{k=1}^{d} \exp\left(\sum_{q=1}^{p} h_q v_{qj}\right)}$ } independent of position i
- As always for softmax loss is cross-entropy:

$$L = -\sum_{i=1}^{m} \sum_{j=1}^{d} y_{ij} \log \hat{y}_{ij}$$

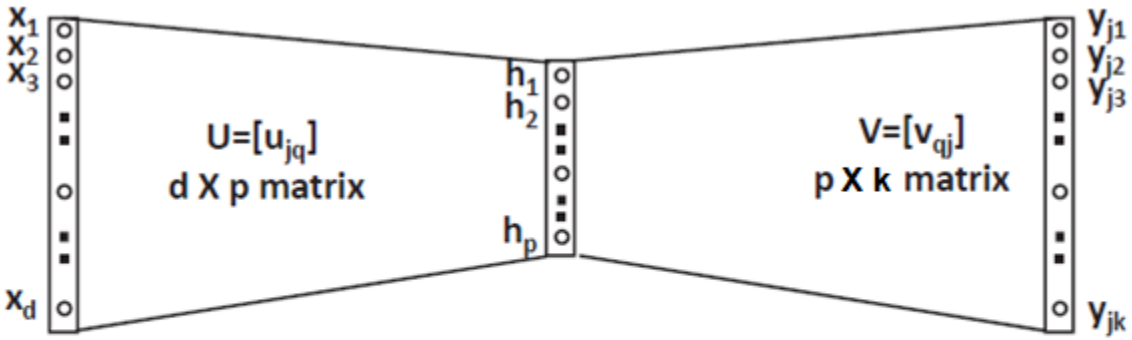
- Updates for a the input word r are computed by backpropagation; can be easily expressed in terms of signed error $\epsilon_{ij} = y_{ij} - \hat{y}_{ij}$ where

$$y_{ij} = \begin{cases} 1 \text{ if word } j \text{ occured at position } i \\ 0 \quad \text{otherwise} \end{cases}$$ as

$\vec{u}_r = \vec{u}_r + \alpha \sum_{j=1}^{d} (\sum_{i=1}^{m} \epsilon_{ij}) \vec{v}_j$ and $\vec{v}_j = \vec{v}_j + \alpha (\sum_{i=1}^{m} \epsilon_{ij}) \vec{h}$ where α is learning hyperparameter

- The training examples of context-target pairs are presented one by one, and the weights are trained to convergence

Skip-gram Architecture with Negative Sampling



MINIBATCH THE m k-DIMENSIONAL OUTPUT VECTORS IN EACH CONTEXT WINDOW DURING STOCHASTIC GRADIENT DESCENT. THE SHOWN OUTPUTS y_{ji} CORRESPOND TO THE jth OF m OUTPUTS.

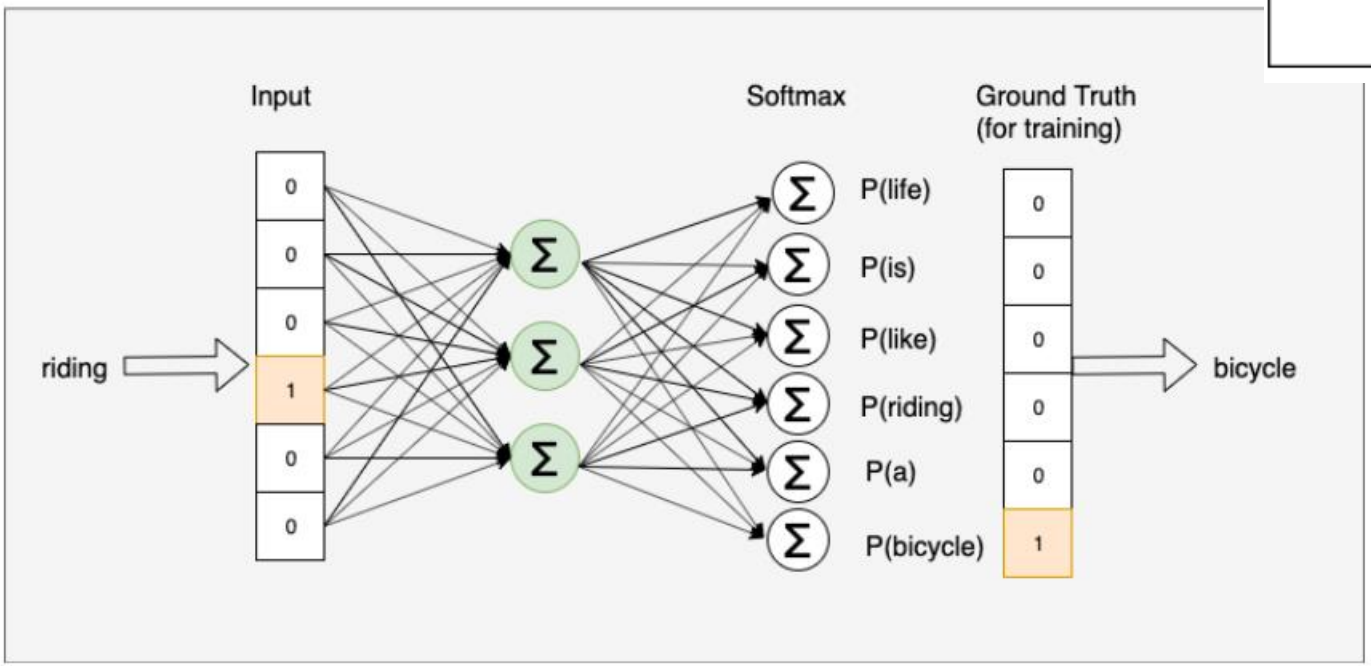
- Gradient descent steps for each instance are proportional to $d \Rightarrow$ Expensive. To improve instead treat the problems as guessing whether the word is present or absent in context
  - Change the softmax layer into sigmoid layer
- Instead of all words use the words present in context and sample words not present
  - For each position of the d outputs, keep the positive output and sample k out of the remaining $d - 1$ (with cross-entropy loss).

Example

life	is	like	riding	a	bicycle

- Looking for skipgram context for riding with $t = 2, m = 2t$ $(+1 \text{ for riding})$
- Target pairs are as in the table

Context	Target
riding	is
riding	like
riding	a
riding	bicycle



Skip-gram Architecture with Negative Sampling

- Since for each word in context we generate k negative samples total size of negative samples is $k \cdot m$.
- How to generate negative sample? Rank the words by frequency of occurrence and sample each word with probability that is equal to relative frequency of the word in the corpus (set of all documents) until you get k words
- For a word w_i that does not occur in this window compute the probability of a word in the thesaurus set

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n} f(w_j)}$$

Where $f(w)$ is the word frequency assigned to a word in thesaurus

- Sample words from a probability distribution given by above formula until we get k words for each given word

Skip-gram Architecture with Negative Sampling

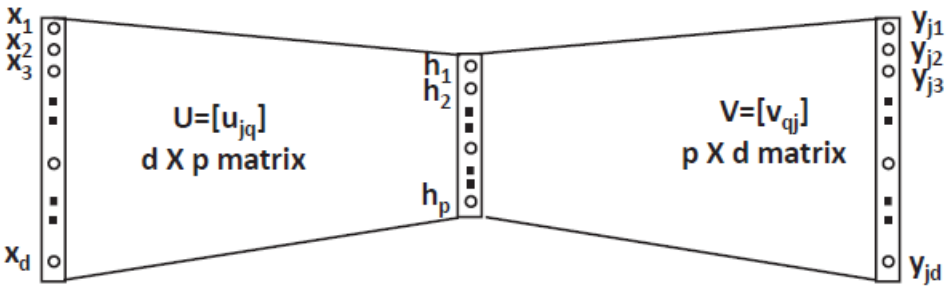
- Then the loss function maximizing log likelihood of present context is

$$L = -\sum_{(i,j)\in\mathcal{P}} \log\left(P\left[X_{i,j} = 1\right]\right) - \sum_{(i,j)\in\mathcal{N}} \log\left(P\left[X_{i,j} = 0\right]\right)$$

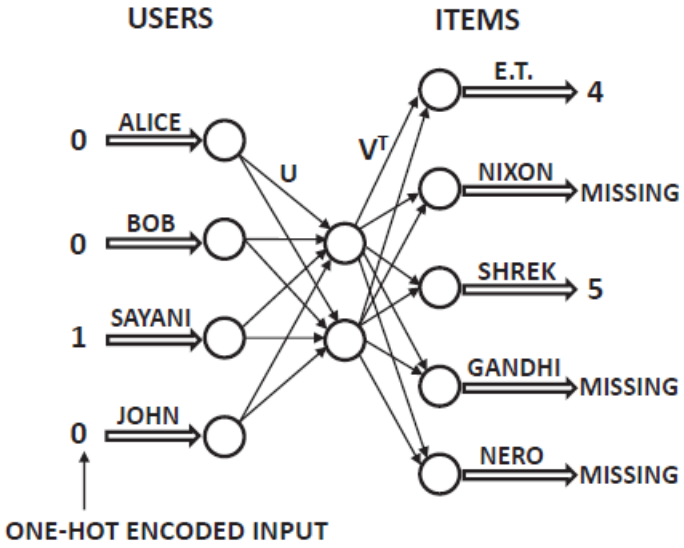
$$= -\sum_{(i,j)\in\mathcal{P}} \log\left(\frac{1}{1+\exp(-\vec{u}_i\cdot\vec{v}_j)}\right) - \sum_{(i,j)\in\mathcal{N}} \log\left(\frac{1}{1+\exp(\vec{u}_i\cdot\vec{v}_j)}\right)$$

where $\mathcal{P}$ is the set of words present in context and $\mathcal{N}$ is the set of negatively sampled words.

Can You See Similarity?



THE VAST MAJORITY OF ZERO OUTPUTS ARE MISSING (NEGATIVE SAMPLING)



- Main difference: sigmoid output layer with binary cross-entropy loss

Word2vec as Matrix Factorizations

- In the book:
  - Equivalence of for Skip Gram w/Negative Sampling (SGNS) to (weighted) logistic matrix factorization (LMF)
  - Gradient descent for SGNS (actually for LMF)
  - Equivalence of Skip-Gram without NS to multinomial matrix factorization (MMF)

Reading

- Ch. 2.5.1.4, 2.5.2 - 2.5.3, 2.5.7, 2.6